**LAB#3**

**ONE-TIME PAD CIPHER**

**Introduction:** one time pad cipher is the improvement of vernam cipher, proposed by an army signal crop officer joseph mauborgne.

**Description:** It is an only algorithm that is unbreakable (completely secure). It is a method of encrypting alphabetic plain text. It is one substitution technique that convert plain text into cipher text. In this mechanism, we assign a number to each character of the plain text.

The two requirements for the One-Time pad are

1. The key should be**randomly generated as long as the size of the message**.
2. The key is to be used to encrypt and decrypt **a single message**, and **then it is discarded.**

**Advantages:**

1. It is one of the most unbreakable in cryptography system
2. It represent the message in the sequence of 0s and 1s.
3. Keys is generated by using random series of 0s and 1s.
4. Keys is not reusable.

**TASK 1:**

**IMPLEMENTATION:**

**Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>One-Time Pad Cipher</title>

</head>

<body>

<h1>One-Time Pad Cipher</h1>

<label for="plain-text">Enter text:</label>

<input type="text" id="plain-text" placeholder="Enter text">

<button onclick="applyCipher()">Encrypt & Decrypt</button>

<p><strong>Generated Key:</strong> <span id="key"></span></p>

<p><strong>Cipher Text:</strong> <span id="cipher-text"></span></p>

<p><strong>Decrypted Text:</strong> <span id="decrypted-text"></span></p>

<script>

class OneTimePadCipher {

constructor() {

this.value = { 'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9, 'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25 };

this.alphabets = "abcdefghijklmnopqrstuvwxyz";

this.key = "";

this.cipher\_text = "";

this.plain\_text = "";

}

generateRandomKey = () => {

this.key = "";

for (let i = 0; i < this.plain\_text.length; i++) {

this.key += this.alphabets[Math.floor(Math.random() \* this.alphabets.length)];

}

}

toBinaryString = (num) => {

return num.toString(2).padStart(5, "0");

}

encryption = () => {

this.cipher\_text = "";

for (let i = 0; i < this.plain\_text.length; i++) {

let plain\_text\_num = this.value[this.plain\_text[i]];

let key\_num = this.value[this.key[i]];

plain\_text\_num = this.toBinaryString(plain\_text\_num);

key\_num = this.toBinaryString(key\_num);

let setter = "";

for (let j = 0; j < plain\_text\_num.length; j++) {

setter += (plain\_text\_num[j] === key\_num[j]) ? '0' : '1';

}

let decimal = parseInt(setter, 2);

this.cipher\_text += Object.keys(this.value).find(key => this.value[key] === decimal);

}

return this.cipher\_text;

}

decryption = () => {

let decrypted\_text = "";

for (let i = 0; i < this.cipher\_text.length; i++) {

let cipher\_text\_num = this.value[this.cipher\_text[i]];

let key\_num = this.value[this.key[i]];

cipher\_text\_num = this.toBinaryString(cipher\_text\_num);

key\_num = this.toBinaryString(key\_num);

let setter = "";

for (let j = 0; j < cipher\_text\_num.length; j++) {

setter += (cipher\_text\_num[j] === key\_num[j]) ? '0' : '1';

}

let decimal = parseInt(setter, 2);

decrypted\_text += Object.keys(this.value).find(key => this.value[key] === decimal);

}

return decrypted\_text;

}

}

function applyCipher() {

const textInput = document.getElementById("plain-text").value.toLowerCase();

if (textInput === "") {

alert("Please enter some text.");

return;

}

const cipher = new OneTimePadCipher();

cipher.plain\_text = textInput;

cipher.generateRandomKey();

const cipherText = cipher.encryption();

const decryptedText = cipher.decryption();

document.getElementById("key").textContent = cipher.key;

document.getElementById("cipher-text").textContent = cipherText;

document.getElementById("decrypted-text").textContent = decryptedText;

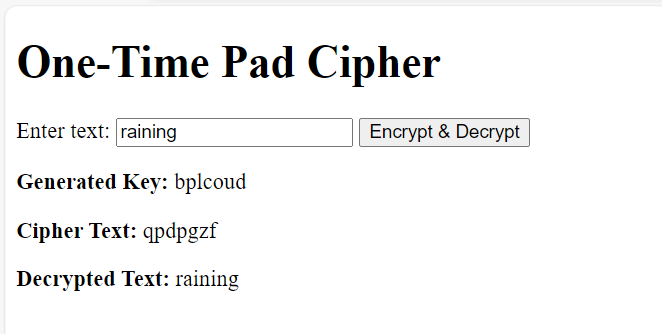
}

</script>

</body>

</html>

**OUTPUT/UI:**

****

**TASK 2**

**IMPLEMENTATION:**

**Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>One-Time Pad Cipher</title>

</head>

<body>

<h1>One-Time Pad Cipher</h1>

<label for="plain-text">Enter text:</label>

<input type="text" id="plain-text" placeholder="Enter text">

<button onclick="applyCipher()">Encrypt & Decrypt</button>

<p><strong>Generated Key:</strong> <span id="key"></span></p>

<p><strong>Cipher Text:</strong> <span id="cipher-text"></span></p>

<p><strong>Decrypted Text:</strong> <span id="decrypted-text"></span></p>

<script>

class OneTimePadCipher {

constructor() {

this.value = { 'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9, 'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25 };

this.alphabets = "abcdefghijklmnopqrstuvwxyz";

this.key = "";

this.cipher\_text = "";

this.plain\_text = "";

}

generateRandomKey = () => {

this.key = "";

for (let i = 0; i < this.plain\_text.length; i++) {

if (this.plain\_text[i].match(/[a-z]/)) {

this.key += this.alphabets[Math.floor(Math.random() \* this.alphabets.length)];

} else {

this.key += this.plain\_text[i];

}

}

}

toBinaryString = (num) => {

return num.toString(2).padStart(6, "0");

}

encryption = () => {

this.cipher\_text = "";

for (let i = 0; i < this.plain\_text.length; i++) {

if (this.plain\_text[i].match(/[a-z]/)) {

let plain\_text\_num = this.value[this.plain\_text[i]];

let key\_num = this.value[this.key[i]];

plain\_text\_num = this.toBinaryString(plain\_text\_num);

key\_num = this.toBinaryString(key\_num);

let setter = "";

for (let j = 0; j < plain\_text\_num.length; j++) {

setter += (plain\_text\_num[j] === key\_num[j]) ? '0' : '1';

}

let decimal = parseInt(setter, 2);

this.cipher\_text += Object.keys(this.value).find(key => this.value[key] === decimal);

} else {

this.cipher\_text += this.plain\_text[i];

}

}

return this.cipher\_text;

}

decryption = () => {

let decrypted\_text = "";

for (let i = 0; i < this.cipher\_text.length; i++) {

if (this.cipher\_text[i].match(/[a-z]/)) {

let cipher\_text\_num = this.value[this.cipher\_text[i]];

let key\_num = this.value[this.key[i]];

cipher\_text\_num = this.toBinaryString(cipher\_text\_num);

key\_num = this.toBinaryString(key\_num);

let setter = "";

for (let j = 0; j < cipher\_text\_num.length; j++) {

setter += (cipher\_text\_num[j] === key\_num[j]) ? '0' : '1';

}

let decimal = parseInt(setter, 2);

decrypted\_text += Object.keys(this.value).find(key => this.value[key] === decimal);

} else {

decrypted\_text += this.cipher\_text[i];

}

}

return decrypted\_text;

}

}

function applyCipher() {

const textInput = document.getElementById("plain-text").value.toLowerCase();

if (textInput === "") {

alert("Please enter some text.");

return;

}

const cipher = new OneTimePadCipher();

cipher.plain\_text = textInput;

cipher.generateRandomKey();

const cipherText = cipher.encryption();

const decryptedText = cipher.decryption();

document.getElementById("key").textContent = cipher.key;

document.getElementById("cipher-text").textContent = cipherText;

document.getElementById("decrypted-text").textContent = decryptedText;

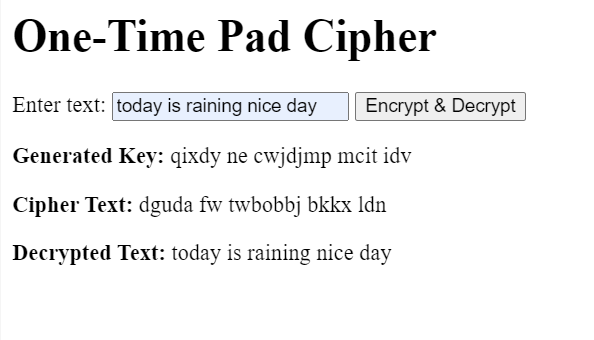
}

</script>

</body>

</html>

**OUTPUT/UI:**

****